Methods for Big Data Analysis
AXA Data Challenge
DatArtists

Olivier Chancé chanceolivier@gmail.com
Céline Lay layceline@gmail.com
Sophia Lazraq sophia.lazraq@gmail.com

January 11, 2017

Introduction

The AXA Data Challenge aimed to predict the number of calls received by AXA France call centers on a per half-hour time slot basis. This prediction should be done 7 days ahead in time based on the history of calls.

In this report, we will detail our approach as well as every step of the pipeline.

1 Data pre-processing and goal understanding

1.1 Goal

First and foremost, we have to understand the data, namely which data we can use to train and which data we have to predict, paying attention to exclusively use the anterior data of the ones we have to predict. Labeled data cover three years, from 01/01/2011 to 12/31/2013. The submission file is a fragmented time series of 10 weeks spreading across 2013, one week per month. Obviously, those weeks don't appear in the training set. Our goal is to predict the calls received during those weeks (namely CSPL_RECEIVED_CALLS) on a 30 minutes slot basis and by ASS_ASSIGNEMENT, only using past values.

1.2 Data pre-processing

Once loaded, we visualize the whole dataset, however since the submission file only contains some features, namely DATE and ASS_ASSIGNEMENT, we keep only those features from the dataset as well as the goal prediction CSPL_RECEIVED_CALLS.

First of all, we group by DATE and ASS_ASSIGNEMENT and sum over CSPL_RECEIVED_CALLS to get the suitable format corresponding to the one of the submission file. Additionally, we get rid of two ASS_ASSIGNEMENT: 'Evenements', 'Gestion Amex' as both doesn't appear in the submission file.

2 Feature Engineering

2.1 Creating new features

In order to analyze the data, we noticed that we needed to transform the data into an appropriate format especially the feature DATE. We simply remove the useless strings "-", ":" as well as the last 0000. Thus, we get a new feature, "DATE_SIMPLIFY", representing time stamps in half-hour slots that we can rank in ascending order.

Once we transformed DATE to DATE_SIMPLIFY, we can easily create new useful features namely the year, month, day, hour, and minute. Besides, we create new features that tell us whether the day in question is a weekday, a weekend or a holiday since these information influence the number of calls received. In addition to the nature of the day, it is relevant to know whether the slot considered is a night-slot or a dayslot. Indeed there are less calls during nights

Eventually, we 'dummify' the feature weekday, in order to get a binary bin per day (from 0 to 6).

2.2 Analysis

To have a very first idea of the data we are using, we operate simple analysis such as the total number of calls, the number of unique date per ASS_ASSIGNEMENT. We also compute the number of data per ASS_ASSIGNEMENT and we highlight the fact that for the ASS_ASSIGNEMENT that present less data, the

missing data concern mostly weekends and nights (slots from 8PM to 7AM). Analyzing data where those slots where available confirmed that we could replace the missing values by 0.

Since there is nothing better than visualization, we plot some interesting graphs:

- The average number of calls per slot (48 slots) per ASS_ASSIGNEMENT: we deduce a periodicity and notice that, indeed, there are more calls during days than nights and that for most ASS_ASSIGNEMENT, there is a drop during lunch hours.

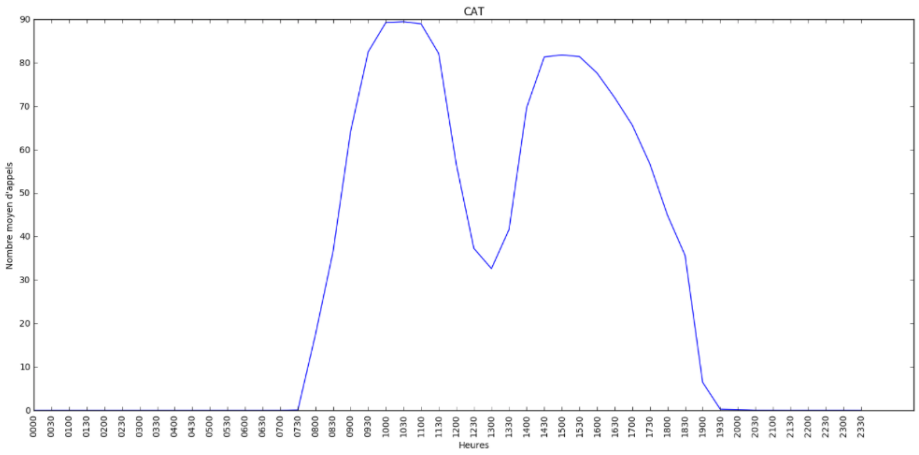

Figure 1: Mean Number of calls per slots for CAT

- The sum of calls per weekday per ASS_ASSIGNEMENT: we notice that the number of calls is particularly high on Mondays since it follows the weekend. The same logic applies with holidays; there is a higher number of calls the day following holidays.

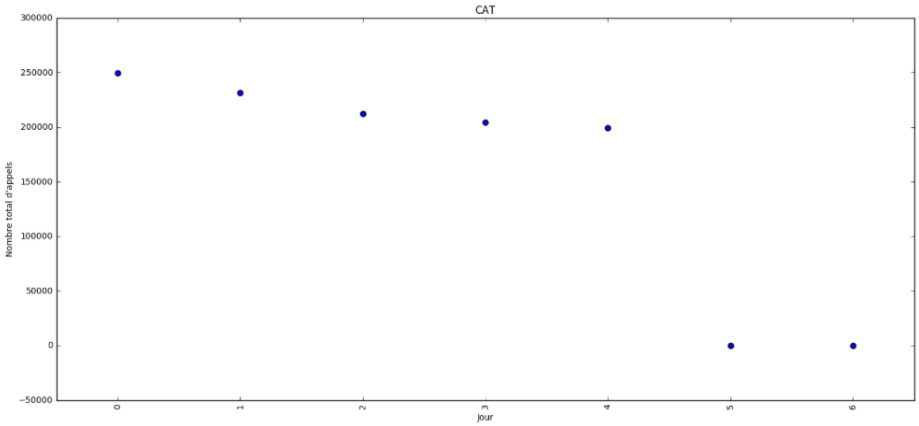

Figure 2: Total Number of calls per day for CAT

- The missing days on the whole period (3 years) per ASS_ASSIGNEMENT for both the local training set and the submission test.
- The percentage of missing days per weekday per ASS_ASSIGNEMENT for the training set: we highlight that most of missing values concern Saturdays and Sundays.
- The number of occurrences of date per slot: we deduce that we have much more data for slots from 7AM to 8PM. From this graph and the previous one, we conclude that missing values are mostly about weekend and night slots.
- The percentage of data per weekday in the submission set: we notice that we have less predictions to do for Saturdays and Sundays (this goes back to the idea that most missing values concern weekends).
- The number of calls over the whole period per ASS_ASSIGNEMENT taking into account weekends and holidays: we realize that there are much less calls during weekends and holidays and that there is an overcompensation the day following the weekend (Mondays) or the day following a holiday.
- This graph shows the influence of the type of day on the number of calls received. We note that on Saturdays, Sundays and public holidays (white dots), the number of calls is null. The day after weekends, namely on Mondays (orange dots) and after holidays (red dots), the number of calls is much more important. The blue dots represent the other, regular days.

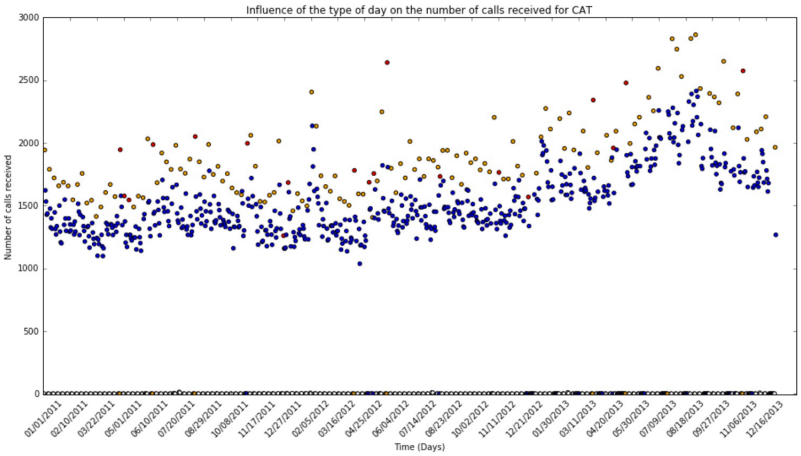

Figure 3: Influence of the type of day on the number of calls received for CAT

All these conclusions from visualization are taken into account when constructing the learning algorithm.

3 Learning Algorithm

3.1 Model

First of all, we constructed a dataframe containing two columns, the first one represents the simplified date feature "DATE_SIMPLIFY" explained previously and the second column corresponds to the goal prediction CSPL_RECEIVED_CALLS. For the second column, if the value is missing, we replace it by 0. Indeed, as we mentionned before, most missing values concern weekends and nights which are slots that have almost no calls in average.

Secondly, we transformed the data to create the useful new features namely:

New Feature Name	Meaning
WEEKEND	1 if date is a weekend 0 otherwise
HOLIDAY	1 if date is a day off 0 otherwise
IS_AFTER_HOLIDAY	1 if (date - 1 day) is a day off 0 otherwise
HOUR_MIN_HL	Ordinal encoding for hourmin, which is 0 for 00h00 1 for 00h30 2 for 01h00 3 for 01h30 ... 47 for 23h30
DAYS_SINCE_BEG	Number of days since 01/01/2011
WEEKDAY_i	0 if Monday 1 if Tuesday ... 6 if Sunday
MONTH	1 if January 2 if February ... 12 if December

Table 1: New Features created

Now that we have the necessary data, we can apply a self training algorithm. It works as follows:

- For each ASS_ASSIGNEMENT, we train a regressor. In our case, it's the Gradient Boosting Regressor.
- In order to train the regressor, we need the list of all weeks for which we have to predict the number of calls.
- For each week_i that we need to predict, we use all the data available before week_i included the predictions we have already done, we do the training over it and then predict the number of calls.

For example, let us put $L = [5, 13, 19, ...]$ the list of weeks for which we need to predict the number of calls, then, for each ASS_ASSIGNEMENT, we do:

1. Take all weeks before week 5, train the regressor and predict the values for week 5.
2. Take all weeks before week 13 including week 5, train the regressor and predict the values for week 13.
3. Take all weeks before week 19 includithe regressor and predict the values for week 19.
4. ...

In pseudo-code, this yields:

```
week_list ← List of weekds in the submission file [week1, week2, ..., week12]
        where weekN is a list of 7 days
class_label_list ← List of all ASS_ASSIGNMENT: ['CAT', 'CMS', ..., 'Telephonie']
data ← All data available, missing values correspond to a date belongind to week_list

for class_label in class_label_list:
        data_cl = data[indexes corresponding to this class_lab]
        for week in week_list:
                //Here, week is a list of 7 days
                data_before_week = data_cl[indexes for data_cl with date < week[0]]
                data_to_predict = data_cl[indexes for data_cl with date included in week]

                regressor = GradientBoostingRegressor()
                regressor.fit(data_before_week)
                data[for this class_lab and this week] = regressor.predict(data_to_predict)

// Data has no more missing values. We can submit our prediction
for row in submission_file:
        date_submission = row[DATE]
        class_submission = row[ASS_ASSIGNMENT]
        //Finally, we get the prediction, but we apply an overestimation coefficient before
        prediction = overestimation_coef*data[data_submission, class_submission]
```

3.2 Model selection

The model we chose has a panel of parameters that we had to select. We tuned these parameters using cross validation:

- Regressor: The most promising models were GradientBoostingRegressor, RandomForestRegressor and BaggingRegressor. However the GradientBoostingRegressor eventually yielded the best performance.
- Parameters: In order to select the parameters that yield the best score, we cross-validated various combination of them and eventually selected the ones described in Table 1.
- Number of trees in the ensemble model: Adding more trees can't hurt the score of the model or overfit it. It only increases the training time. We noticed that after 1000 trees, performance didn't increase anymore, so we chose this number.
- Overestimation coefficient: GradientBoostingRegressor is not trained by minimizing the loss we finally want to minimize so to get better results, we have to apply an overestimation coefficient. The best coefficient we found is 1.25. As our model becomes more precise, the overestimation coefficient decreases.

In case of time series, the regular cross validation is not suitable, as one would use the future to predict the past. Instead, we used the following schema:

- We picked 12 weeks in 2012: first week (01/01 to 01/07), 2nd week (02/07 to 02/14), 3rd week (03/14 to 03/21), 4th week (04/21 tp 04/28), 5th week (05/01 to 05/07), ... until December. In fact, the weeks we pick are similar to the ones in the submission file except they correspond to the previous year. Those weeks will be used as the validation set.
- Then, we apply the same method as the one explained above to predict the calls received for a set of weeks, except that all data are available for 2012, so we can use this validation to tune all the parameters described above .

4 Results

After 30 trials, we improved our score from 0.86 to 0.2804 on the public leaderboard. Our last score matches the model we described throughout this report.

5 Discussion and possible improvement

Some methods seemed to add some enhancement to the model but didn't work well on the leaderboard. For example, adding a column that returns the mean of calls received on the 7 previous days for the ASS_ASSIGNMENT significantly improved our score locally but eventually failed to improve the leaderboard score.

We can see the following as possible improvements we could make:

- Autoregressive methods seem to be promising methods that could help improve our score, especially for ASS_ASSIGNMENT that gets a mediocre score when predicting for CAT, Services or Téléphonie.
- Determine a different overestimation coefficient for each ASS assignement.

Conclusion

In order to tackle the challenge, we meticulously followed the steps of the pipeline since it suggests a good methodology to solve data problems. The pre-processing step and the analysis were crucial. Indeed, both these steps led us to highlight some important behaviors of the data, using the different behaviors to construct new features drastically improved the score.

As for the chosen model, we tried many of them before selecting the one that fits the best our data. Cross-validation especially helped us to tune the parameters and hence determine our final solution.

As for most of challenges, it is certainly still possible to get a better score by deepening more in the specificities of the dataset and using more complex models.